Linear Models

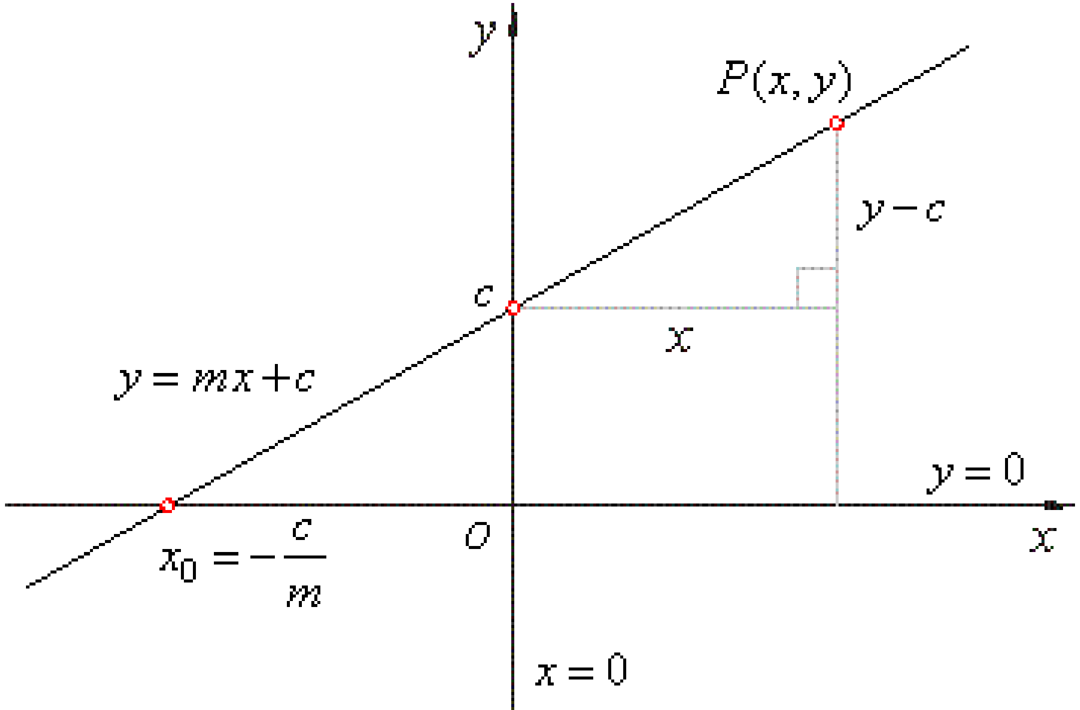
Linear Regression/Classification

- Regression is concerned with continuous data (i.e. real numbers)
- Ex: stock market predictions, houses prices, weather temperatures, …
- Classification, is to discriminate between linearly separable data with linear model (linear, plan, …)

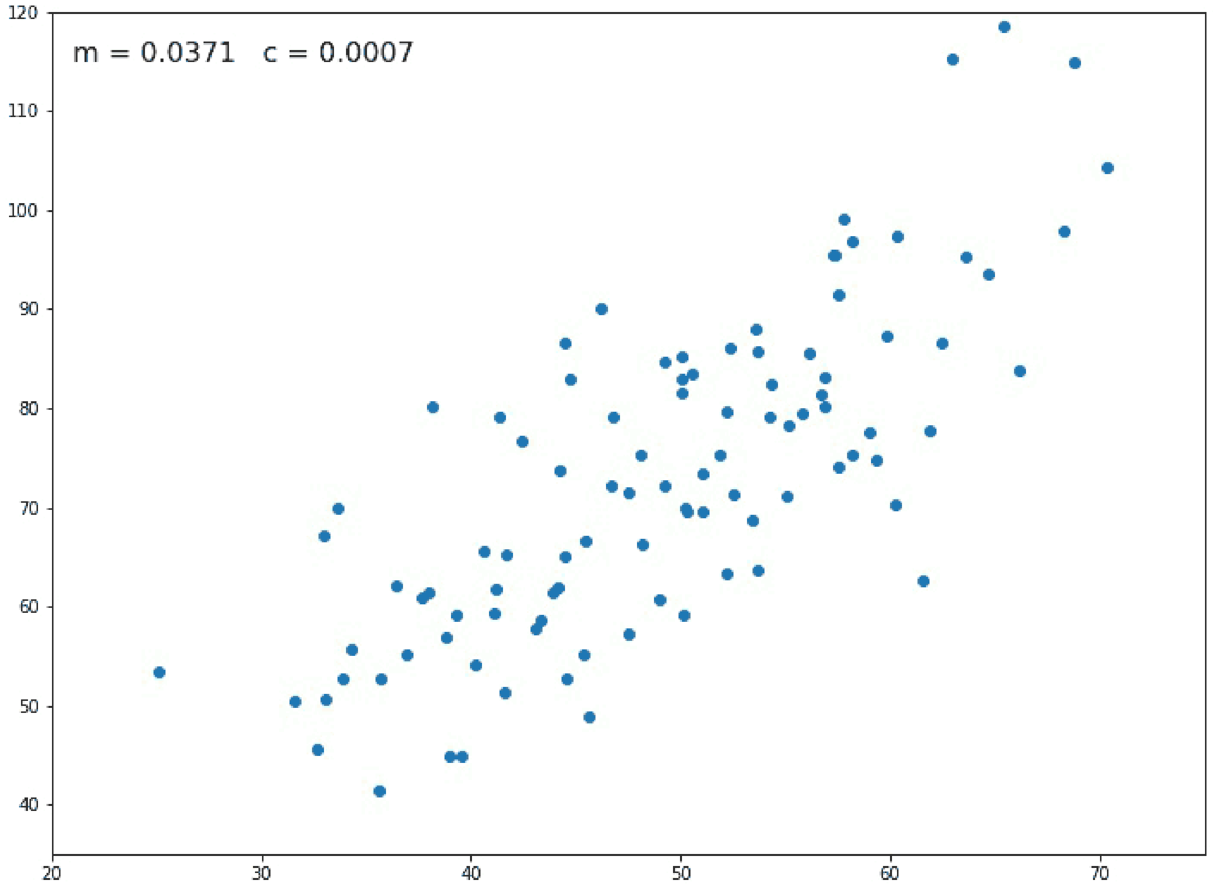
Linear regression

- Linear regression is a linear approach for modelling the relationship between a dependent variable and one or more independent variables. Let **X** be the independent variable and **Y** be the dependent variable. We will define a linear relationship between these two variables as follows:

$$Y = wX + b$$



The values of *m* (or ‘*w*’) and *c* (or ‘*b*’) are updated at each iteration to get the optimal solution



Linear Regression: In Higher Dimensions

- Given training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
- Fit each training example $(\mathbf{x}_i, y_i)$ using the linear model

$$y_i = b + \mathbf{w}^T \mathbf{x}_i$$

- A bit of notation abuse: write $\mathbf{w} = [b, \mathbf{w}]$, write $\mathbf{x}_i = [1, \mathbf{x}_i]$

$$y_i = \mathbf{w}^T \mathbf{x}_i$$

- Switching to matrix notation, the relationship becomes: $\mathbf{Y} = \mathbf{X}\mathbf{w}$

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \mathbf{X} = \begin{pmatrix} 1 & \mathbf{x}_1 \\ \vdots & \vdots \\ 1 & \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1D} \\ \vdots & & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{ND} \end{pmatrix}, \mathbf{w} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$$

- $\mathbf{Y}$: $N \times 1$, $\mathbf{X}$: $N \times (D+1)$, $\mathbf{w}$: $(D+1) \times 1$

Linear Regression: The Objective Function

- Parameter $\mathbf{w}$ that satisfies $y_i = \mathbf{w}^T\mathbf{x}_i$ *exactly* for each i may not exist
- So we look for the closest approximation
- Specifically, $\mathbf{w}$ that minimizes the following sum-of-squared-differences between the truth (y_i) and the predictions ($\mathbf{w}^T\mathbf{x}_i$), just as we did for the one-dimensional case:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

- Following the matrix notation, we can write the above as:

$$E(\mathbf{w}) = \frac{1}{2}(\mathbf{Y} - \mathbf{X}\mathbf{w})^T(\mathbf{Y} - \mathbf{X}\mathbf{w})$$

Linear Regression: Least-Squares Solution

- Taking derivative w.r.t **w**, and equating to zero, we get

$$\begin{aligned} \nabla E(\mathbf{w}) &= -\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\mathbf{w}) = 0 \\ \Longrightarrow \mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{Y} \end{aligned}$$

- Taking inverse on both sides, we get the solution

$$\boxed{\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}}$$

- The above is also called the least-squares solution (since we minimized a sum-of-squared-differences objective)

- **Note:** The same solution holds even if the responses are vector-valued (assume K responses per input)
  - **Y** will be an $N \times K$ matrix (assuming K responses per input)
  - **w** will be a $D \times K$ matrix (k-th column is the weight vector for the k-th response variable)

$$L(w) = \frac{1}{2}(XW - Y)^T(XW - Y)$$

$\underline{\min_{w} L(w) : \text{obj}}$

$$\frac{1}{2}\left[(XW - Y)^T X + X^T(XW - Y)\right] = 0$$

$$X^T(XW - Y) = 0$$

$$X^T X W = X^T Y \quad [* \, (X^T X)^{-1}]$$

$$\boxed{W = (X^T X)^{-1} X^T Y}$$

Cost/Loss Function

- $L(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \rightarrow \frac{1}{2m}\sum_{i=1}^{m}((w_i.x_i + b_i) - y_i)^2$

- We want it to be the minimum to get best parameters that describe the model that best fit the data.
- Two methods: Normal function & Gradient Decent Algorithm.
- So, differentiate the loss w.r.t weights and equate to zero.

- $$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{w}\right)^2 \equiv \sum_{i=1}^{N} e_i^2$$
- Differentiate loss w.r.t W → $$\sum_{i=1}^{N} \boldsymbol{x}_i \left(y_i - \boldsymbol{x}_i^T \hat{\boldsymbol{w}}\right) = 0$$
- $$\left(\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T\right) \hat{\boldsymbol{w}} = \sum_{i=1}^{N} (\boldsymbol{x}_i y_i)$$

For the sake of mathematical formulation let us define

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{3.42}$$

That is, X is an $N \times l$ matrix whose rows are the available training feature vectors, and $\boldsymbol{y}$ is a vector consisting of the corresponding desired responses. Then $\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T = X^T X$ and also $\sum_{i=1}^{N} \boldsymbol{x}_i y_i = X^T \boldsymbol{y}$. Hence, (3.41) can now be

- $(X^T X)\hat{\boldsymbol{w}} = X^T \boldsymbol{y}$

- $\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y}$

$$\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \Rightarrow \frac{1}{2m}\sum_{i=1}^{m}\left(w_i x_i + b_i - y_i\right)^2$$

W is a vector when x is an nD features

In matrix form: $L = \frac{1}{2m}\sum_{i=1}^{m}\left[(w_1 x_1 + w_2 x_2 \cdots w_n x_n + b) - y_i\right]^2$

$$= \frac{1}{2m}\sum_{i=1}^{m}\left(w_i^T X_i - y_i\right)^2$$

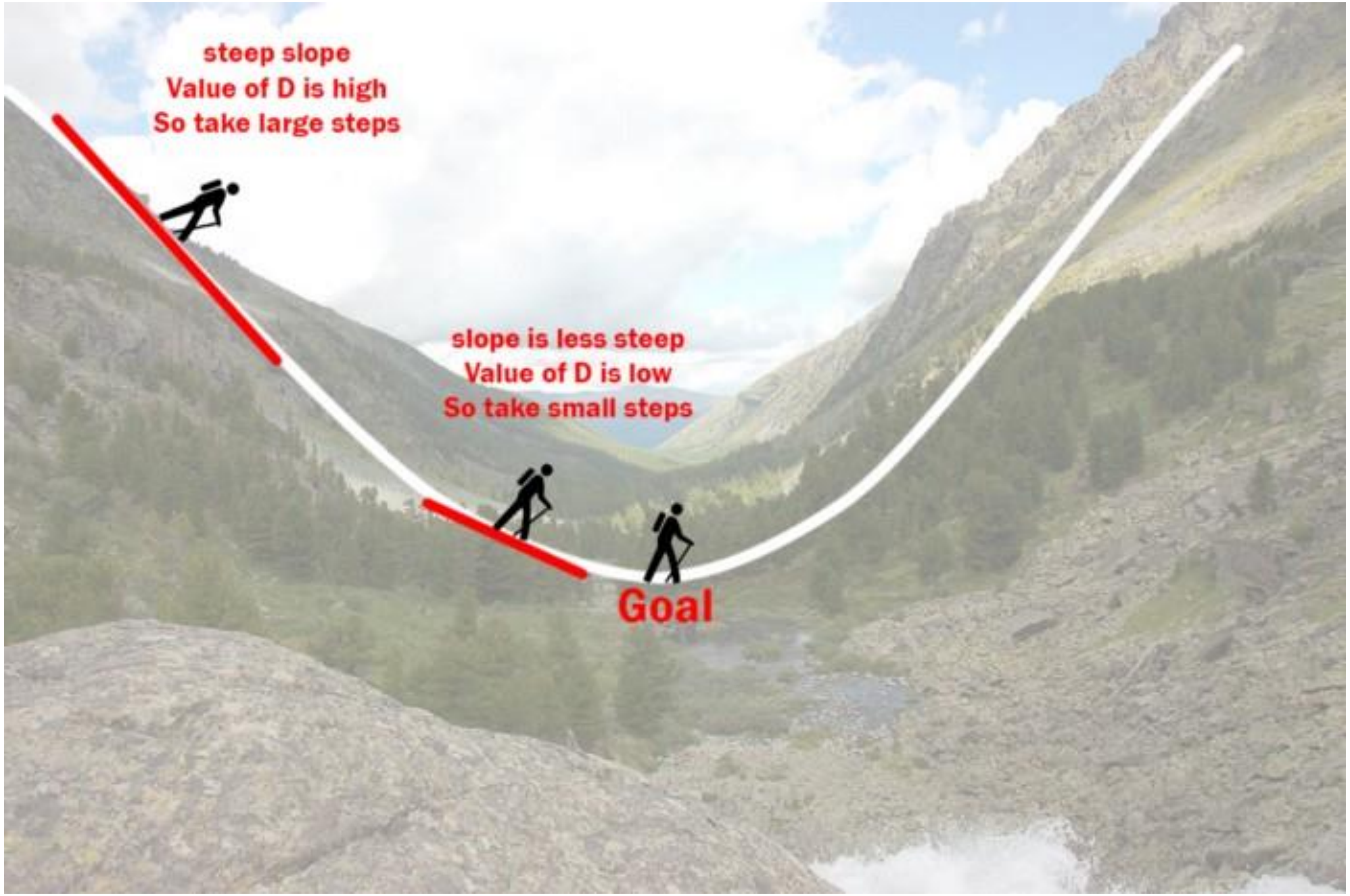
$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} \qquad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \\ b \end{bmatrix}$$

Design Matrix

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} & 1 \\ x_{21} & x_{22} & \cdots & x_{2n} & 1 \\ \vdots & & & & \\ x_{m1} & x_{m2} & \cdots & x_{mn} & 1 \end{bmatrix}_{m \times (n+1)} \quad W = \begin{bmatrix} w_1 & \cdots & w_{m1} \\ w_2 & & \vdots \\ w_3 & \cdots & \vdots \\ \vdots & & \vdots \\ w_n & & w_{mn} \\ b_1 & & b_m \end{bmatrix}_{(n+1)m} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{m \times 1}$$

$$L(w) = \frac{1}{2}(XW - Y)^T(XW - Y)$$

Gradient Descent



Gradient Descent

- $W = w - \alpha \frac{\partial L(w)}{\partial w}$
- $\frac{\partial L(w)}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} x_i (w_i^T x_i - y_i)$

$$= X^T(XW - Y)$$

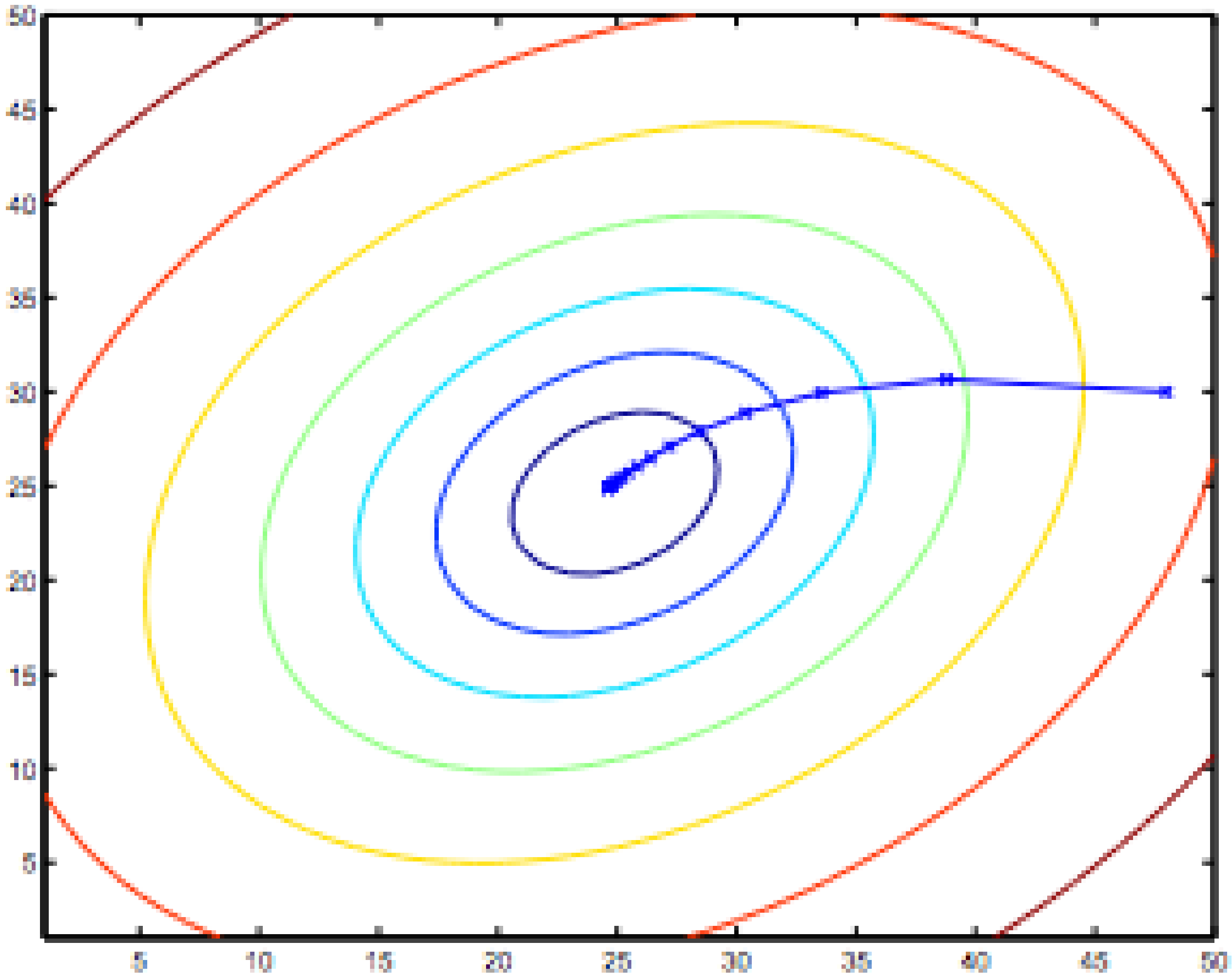
- Solve iteratively
- α ➔ is the learning rate (the step to walk with on the loss curve)

$$L(w) = \frac{1}{2m}\sum_{i=1}^{m}(w_i^T x_i - y_i)^2$$

$$L(w) = \frac{1}{2}(XW - Y)^T(XW - Y)$$

- **If α is too large, the gradient decent can overshoot the minimum. It may fail to converge.**
- **If α is too small, the gradient decent will be too small But it will reach the minimum.**

Gradient Descent steps on the weights to get the global minimum



Notes

- Features normalization is a good practice as it makes gradient descent converges faster, avoid feature bias by setting all features in the same range (0 → 1)

Linear Assignment

- Generate 1000 linear samples that follow the following equation:
  - $Y = 5X_1 + 3X_2 + 1.5X_3 + 6$
  - Hint: generate the Xs randomly.
- Use built-in function to split the data to (Train & Test)
- **Implement** **<u>Loss</u>** & **<u>gradientDescent</u>** functions
- **Print** the final weights & Accuracy
- The only built-in function allowed in this assignment is the split fn. It is not allowed to use any other built-in function in the code.
- You have to implement the gradient descent to update the weights using the equation included in this lab (Don't use any other functions based on google search) (You will not get its grade)

Assignment in details

For the assignment:

$Y = 5X_1 + 3X_2 + 1.5X_3 + 6$

1)

it means that the initialization of the weights are w1 = 5, w2=3, w3=1.5, w4 = 6 or make it as vector

W = (5, 3, 1.5, 6)

X_1 = generate vector of 1000 random number

X_2 = generate vector of 1000 random number

X_3 = generate vector of 1000 random number

Then calculate Y using the above equation

Now you have your dataset

2) Use the built in function to split the data to (Train & Test) or use the split function you did in the first python assignment.

Assignment in details

3) initialize n_iter as number of iterations, initialize learning rate with any random number (for example n_iter=1000, lr=0.01)
X_b is the xTrain after adding ones for the bias

```
theta,cost_history,theta_history = gradientDescent(X_b, YTrain, weights, LR = 0.01, iterations=100)
```

in this function, you will implement the gradient descent equation of updating the weights and print the updated weights

4)
implement loss/cost function

```
costFn(weights, X, Y)
```

and print the loss

5)
Calculate the accuracy between
-the actual output Y which calculated in step 1 (using initialized weights) of the test part and
-predicted output using the calculated weights of step 3 multiplied by the test data